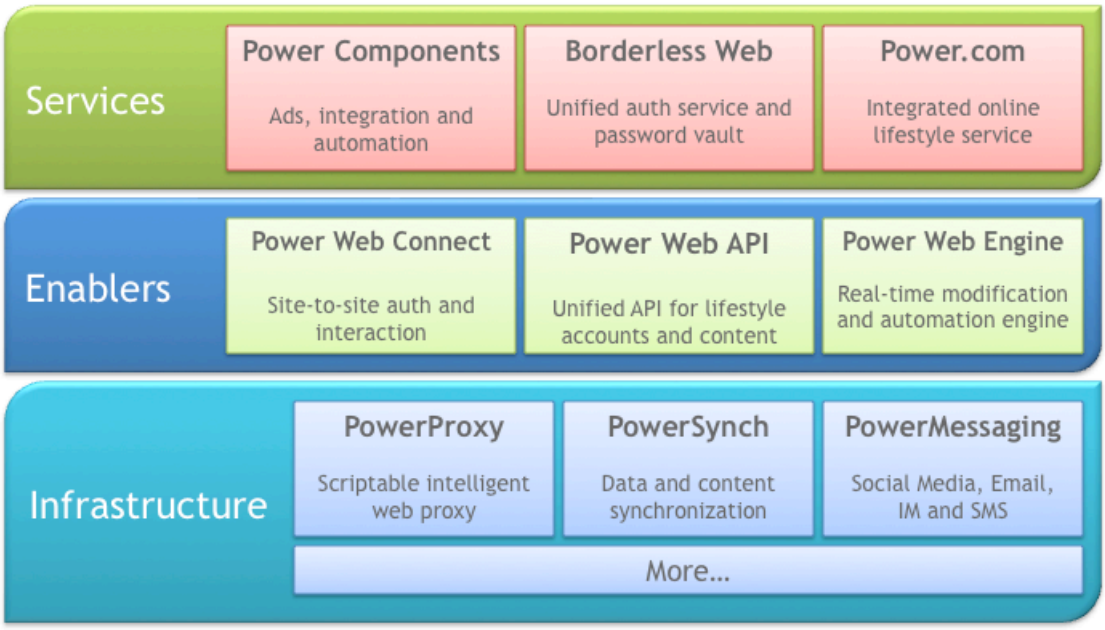
EXHIBIT 32

Power Technology Overview

Power has created the foundation of an open platform that will allow developers to create thousands of value-added applications on top of any web site. Using Power's groundbreaking technology, developers can extend and integrate ecommerce site (e.g. eBay, Amazon and WalMart.com) with social networks (e.g. Facebook, MySpace and LinkedIn) with financial services sites (e.g. Capital One, Bank of America, eTrade). The possibilities are limitless.

Following diagram shows the key components of our technology.



Infrastructure Layer: At the heart of our infrastructure is PowerProxy, the intelligent programmable web proxy (also referred to as web bot) that can interact with any site, extract information, submit forms, upload files, etc. on behalf of the user. PowerSynch component handles the complex logic for aggregating and synchronizing content and data. PowerMessaging aggregates messages, including email, social network messages, instant messages and SMS.

Enablers Layer: This layer provides higher-level programmatic access to infrastructure. Power Web Connect provides the standard interface for building adapters to connect to any site through PowerProxy or third-party APIs. Power Web API provides a high order unified API and schemata for accessing life-style services. For example, Power Web API will offer a universal API call for accessing a user's friends, regardless of social network. This approach frees developers from learning proprietary APIs for each provider.

Power API is an HTTP REST based API, similar to APIs from Google, Flickr, Yahoo and Twitter. REST APIs are compatible with all major programming languages and platforms, including Java, .NET, PHP, Ruby, Python, PERL, etc.

PowerScript, which will evolve into Open Web Markup Language, is a markup language similar to HTML, ColdFusion and Facebook Markup Language (FBML). Power API is a higher order API. A developer calls Power API in much the same way they call Facebook API. Power's Web Connect adapter translates the API call to the provider-specific APIs, aggregates the response, and returns the information to the caller in a standard format.

Power Web Engine provides the ability to do mashups by modifying sites in real-time, layering value-added services over existing web sites. For example, a developer could add a new option in YouTube to instantly share a video with friends. Web Engine also offers a robust execution engine for developing powerful macros to automate user actions. For example, a developer could build a macro that responds to an SMS messages and updates a status message on all major social networks without much coding.

Services Layer: Power Components are revenue-enabled embeddable web components that can be added to any site by the site owner, webmaster, developer, etc. by simply registering to use it. Power Components include Power Ads, our social advertisement marketplace that offers unparalleled value by enabling one-click registration of new accounts. It also includes pay-per-transaction components that support one-click account import, application install, friends invite, and more.

In addition, Power Components include ad-based components that perform complex operations like photos synch, status synch, profile synch, etc. available free of charge to developers.

Borderless Web is our universal authentication and authorization service. It is an independent data store and service that securely authenticates a user to any site, maintains a vault of user credentials and active sessions.

Power.com is our consumer-facing online service that gives users a unified access to all major online services. Today, Power.com integrates all major social networks to give users a single place to connect with all their friends. The next version, Power.com 2.0, will go beyond social networks to include the top 25 service sites covering all major verticals, including careers, financial services, dating, travel, entertainment and more. Power.com 2.0 will also include universal address book and universal communicator.

Power Architecture Overview

Following diagram illustrates the core technology architecture of Power.com and related infrastructure. This is a logical view of the hardware and software components.

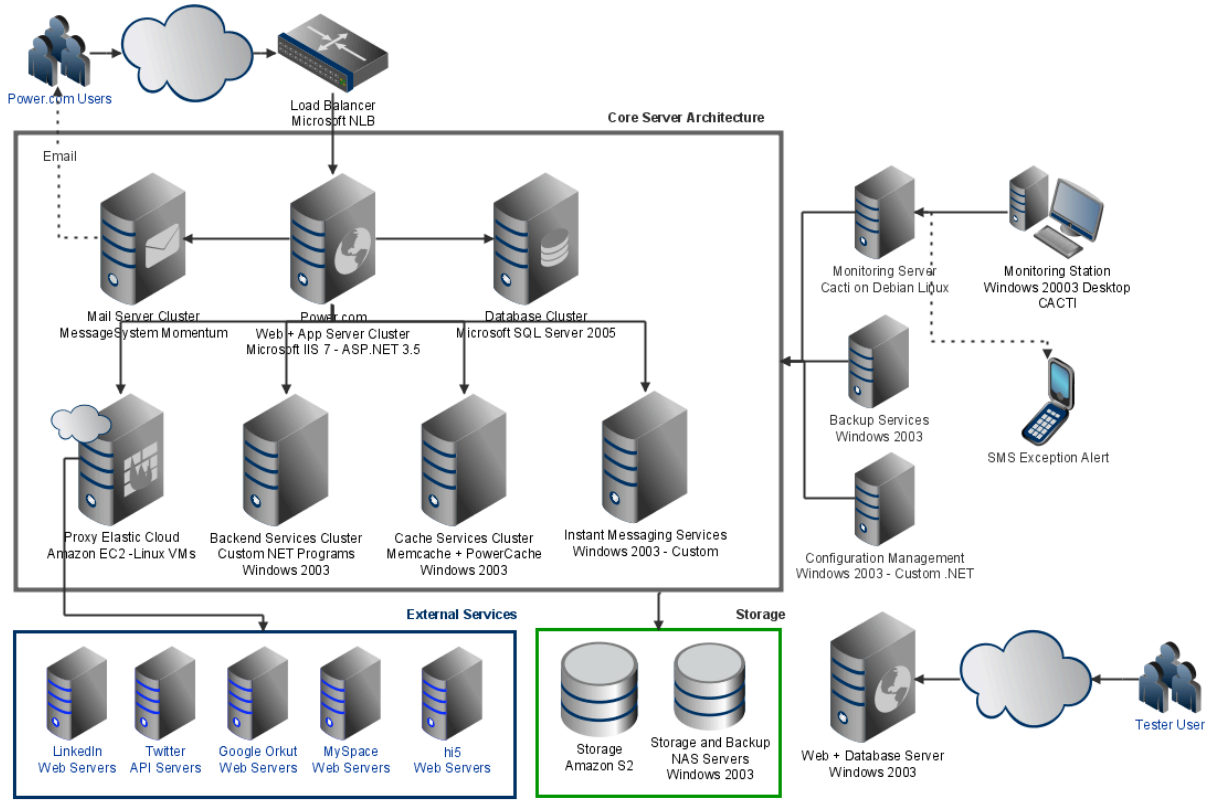


Figure 1 - Power.com System Architecture

Description of key components follows:

Core Server Architecture

This group of servers and applications perform the core Presentation and Business Logic for Power.com.

- Load Balancer: Power.com users access the system via the Microsoft NLB load balancer. The load balancer routes incoming traffic to the appropriate Power.com Web + App Server from the cluster.
- Power.com Web + App Servers: These servers handle the incoming HTTP traffic, and execute the appropriate .NET compiled code. Power.com core web site is built with C# using Microsoft ASP.NET 3.5 and custom classes built using Microsoft Application Blocks design patterns. The .NET application accesses data from the SQL Server databases, Memcached servers, and Proxy servers. Email is sent via the Mail Server Cluster. Asynchronous tasks are passed onto the Backend Services Cluster.

- Database Cluster: The database service is provided by optimized Microsoft SQL 2005 instances. Actual data is stored on the NAS Servers, and backed up by Backup Services.
- Mail Server Cluster: These servers run Message System's Momentum solution for reliable outgoing email, with tracking, retry, and intelligent exception handling.
- Proxy Elastic Cloud: These are Linux virtual servers hosted by Amazon's Elastic Cloud Computing service that perform distributed HTTP proxy. Proxy servers are invoked by .NET-based PowerProxy engine that interprets and executes PowerScript language.
- Backend Services Cluster: These Windows-based servers host multiple custom .NET programs that perform scheduled and asynchronous tasks.
- Cache Services Cluster: These servers run Memcache and Power's proprietary caching technology. Cache servers store frequently accessed data in memory to optimize performance.
- Instant Messaging Services: These are dedicated servers running custom .NET code for Power Chat and Power Instant Messenger.

External Services

Power Proxy servers connect to third-party servers, currently LinkedIn, HI5, MySpace, Orkut and Twitter, to access and aggregate information on behalf of the user. Proxy servers connect to HTML interfaces as well as APIs. By using an elastic cloud for proxying, the IP addresses and load is distributed across many small virtual machines that are provisioned dynamically to meet demand.

Storage Services

Low-level data storage is done on Amazon's Simple Storage Service (S3) and Power's own Network Addressed Storage (NAS) infrastructure with redundancy. NAS is backed up with 100% redundancy using NT backup. S3 is automatically backed-up by Amazon as part of the service.

Other Components

- Monitoring Server: This server monitors all the servers using Cacti monitoring service running on Debian Linux. Cacti performs network-level (e.g. Ping, TCP/IP) as well as application-level (e.g. HTTP, NUnit, SMTP) polling. Cacti reports exceptions through exception log, and generates SMS-alerts for critical failures.
- Backup Services: Running NT Backup Service, this server performs scheduled backup of storage infrastructure.
- Configuration Management server: This server hosts configuration information, version controlled using Subversion. It is also responsible for automatically propagating configuration changes across the infrastructure. Configuration

Management server also doubles as automated testing and monitoring service. It periodically executes NUnit based unit-tests for all major PowerScript scripts – logging reports and issuing notifications.

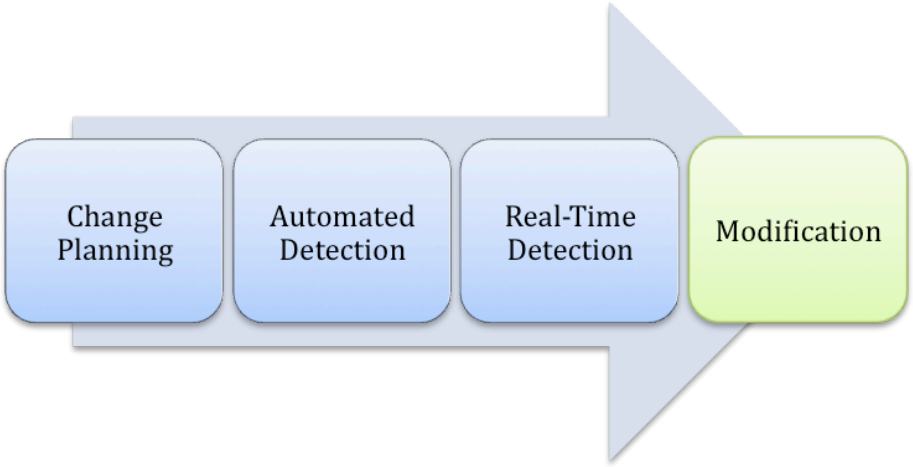
- Monitoring Station: This is a desktop manned by an administrator for proactively monitoring all operations. It also serves as a management console for day-to-day system administration activities.

Test Environment

Power infrastructure also includes a smaller-scale environment that replicates production. It mimics production components, allows developers to host new versions of components, and enables testers perform functional tests.

Exception Handling Mechanisms

Our current exception handling process is designed for internal use at Power.com. The following process describes how changes to source site are detected proactively, and how impact of exceptions is minimized:



1. Change Planning: Power.com analysts follow the source site's blogs, news and updates to learn about upcoming changes. For example, when Google Orkut and MySpace refreshed their web site, they announced 3 months in advance about the upcoming changes. Both sites also provided option to try the beta user interface. This enabled the Power team to proactively track planned changes and build new versions of PowerScripts to be released in conjunction with the source site.

2. Automated Detection: Power.com uses NUnit based automatic execution of test scripts. These scripts are executed 24 hours a day at a 1 to 10 minute interval – adjusted according to release schedules. Following screenshot shows a snapshot

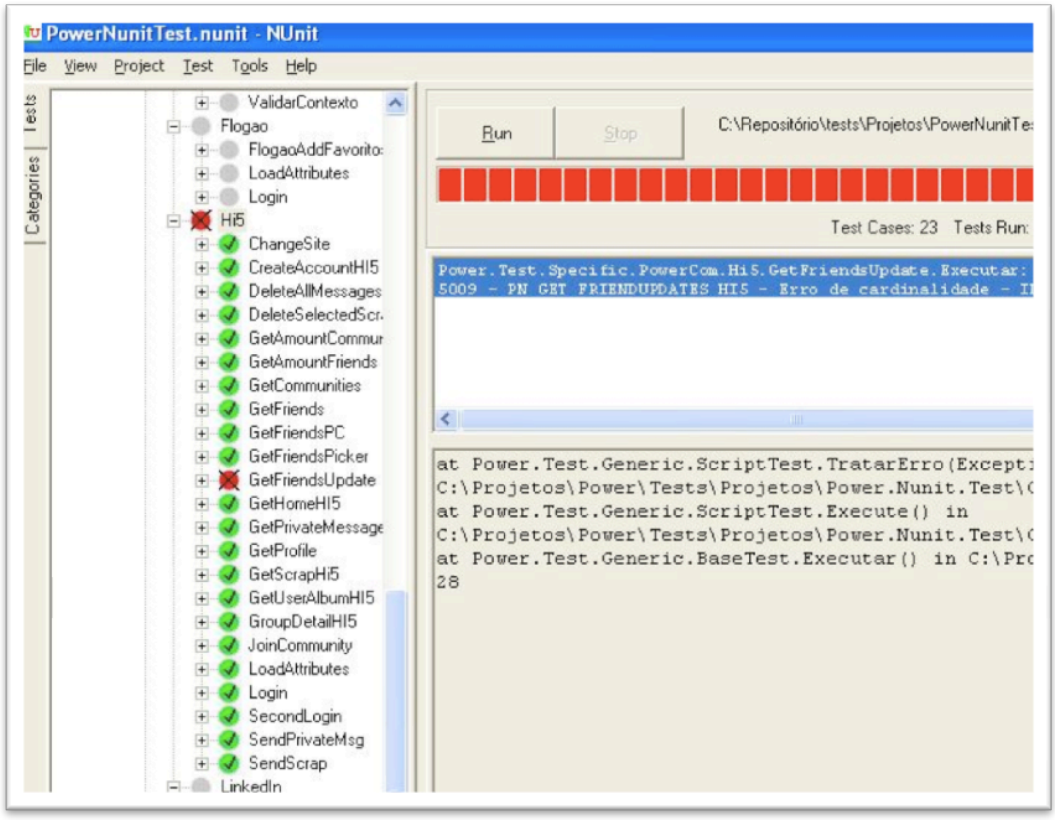


Figure 2 - NUnit PowerScript Detection

report from the NUnit console:

NUnit automated detection will detect all material changes to a source site that causes a PowerScript failure. For example, if hi5 changed the URL of “Friends Updates” page (as shown above), NUnit will detect that while failing to execute that test. The Cacti monitoring station will then report the change based on rules set.

Not all changes will cause script failure. For example, if hi5 were to change the color scheme, or rename button labels (not the internal ID of a button), add new features, etc., this will have no impact on the PowerScript execution or the operation of Power.com.

3. Real-time Detection: For the rare occasions where changes are not detected by the previous two steps, the error may occur while a user is using a certain feature in Power.com. For example, this could occur when a source site change is made without prior notice and also between the time interval of NUnit tests. A high-priority exception is logged in the Windows Event log, and immediately

processed by Cacti. Cacti will alert the operator via email, IM, or SMS based on rules set.

4. Resolution: Planned changes follow the normal development process for resolution. The Power team performs all the standard testing procedures and schedules the deployment according to the release dates of the source site.

   For exception resolution, the exception report provides the operator/programmer a detailed message showing the name of PowerScript file, a list of all dependent scripts, line numbers, list of missing or mismatched elements, date, time, and (when applicable) affected users. PowerScript programmers make the modification or "hot patch", run the updated unit tests, and deploy the change using the configuration management process.

   For complex changes that require more time to resolve, e.g. a complex JavaScript change on source site, the affected feature is disabled in Power.com using an automated configuration script. PowerScript programmers and developers make the code changes, test, deploy and enable the feature as soon as possible. In almost all cases, complex changes are part of a major release that get caught in step 1. Affected users are notified via email, as appropriate and when necessary.

This process has worked very well for Power.com. The team has been able to stay on top of changes and keep real-time detected exceptions to a minimum. We expect the process will evolve, and will be further enhanced and automated as part of the development process of our Open Web Platform.
The following PowerScript source code, which is part of a Google search adaptor, further clarifies the process.

```
1 <processpage>
2   <rules>
3     <rule action="read" param="http://www.google.com/">
4       <var name="formHTML" of-tag="form" text-element="true"/>
5       <var name="qInput" value="#formHTML#" after="&lt;input" before="&gt;" contain="name=q"/>
6     </rule>
7   </rules>
8 </processpage>
```

Figure 3 - PowerScript Code Example

- Line 3 tells PowerProxy to do a "read" on the URL http://www.google.com.
- Line 4 asks PowerProxy to find the HTML form and assign to variable "formHTML." This form contains the text box for users to type in the search criteria.
- Line 5 asks PowerProxy to find an HTML Input element (text box) named "q" inside the variable "formHTML"

Essentially, lines 3 to 5 define the adaptor for processing the Google search page, ready to take input values. This script will execute regardless of how the page looks like to the user, what image is shown for Google's logo, etc. It will work regardless of the default

language selected by the user (English, Portuguese, etc.). As long as the internal element name for the text box, which is expected to be “q”, is not changed, this script will work. However, if a textbox named “q” is not found on the page anymore, this script will fail. In that case, PowerProxy will log an exception with the message, “PowerScript ‘GoogleSearchRead’ failed at line 6; Element containing name=’q’ not find. ” Based on rules defined in Cacti, the appropriate parties will get notified via email, IM or SMS. PowerEngine error reporting is currently being enhanced to add “suggested changes” to PowerScript exceptions. This feature will detect the change and intelligently suggest a fix. In this example scenario, if Google changed the internal name of textbox, PowerEngine exception handler would add “Did you mean ‘htmlForm’?” to the exception report.

The above example illustrates the most basic processing and exception handling functions of PowerScript. There are many advanced options in PowerScript, such as:

- Conditional handling of exceptions: If a named element is not found, find a different element or perform a different action.
- Find elements by sequence instead of name: For example, find the first text box in the Form, regardless of internal name.
- Find an element by matching name within a list of possible names, or partially matching name.